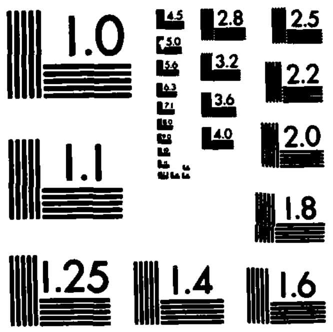
AD-A122 827 A MULTIPROCESS NETWORK LOGIC WITH TEMPORAL AND SPATIAL MOEALITIES REVISED(U) HARVARD UNIV CAMBRIDGE MA AIKEN COMPUTATION LAB J H REIF ET AL. OCT 82 TR-29-82-REV 1/1

UNCLASSIFIED N00014-80-C-0674 F/G 12/1 NL

END

FILMED

DTIC


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Harvard University

Center for Research in Computing Technology

A MULTIPROCESS NETWORK LOGIC WITH TEMPORAL AND SPATIAL MODALITIES

John H. Reif

A. P. Sistla

TR-29-82

August 1982

Revised October 1982

A MULTIPROCESS NETWORK LOGIC WITH TEMPORAL AND SPATIAL MODALITIES*

John Reif
Aravinda Prasad Sistla

Aiken Computation Laboratory
Harvard University
Cambridge, MA 02138

Summary

We introduce a modal logic which can be used to formally reason about synchronous fixed connection multiprocess networks such as of VLSI. Our logic has both *temporal* and *spatial* modal operators. The various temporal modal operators allow us to relate properties of the current state of a given process with properties of succeeding states of the given process. Also, the spatial modal operators allow us to relate properties of the current state of a given process with properties of the current state of neighboring processes. Many interesting properties for multiprocessor networks can be elegantly expressed in our logic. We give examples of the diverse applications of our logic to packet routing firing squad problems, and systolic algorithms.

We show that deciding validity of a formula in our logic is not decidable. However, we show that deciding validity of a proportional formula in our logic with respect to a given finite network is PSPACE-complete. We also investigate the decidability issues of different versions of this logic.

*This work was supported by the National Science Foundation Grants NSF MCS79-21024 and NSF MCS79-08365 and the Office of Naval Research Contract N00014-80-0674.

AD A122827

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Multiprocess Network Logic with Temporal and Spatial Modalities		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-29-82
7. AUTHOR(s) John H. Reif, Aravinda Prasad Sistla		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0674
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University, Cambridge, MA 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Navel Research, 800 North Quincy Street, Arlington, VA 22217		12. REPORT DATE October, 1982
		13. NUMBER OF PAGES 15
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) same as above		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

2

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DTIC
SELECTE
DEC 28 1983
D
S
H

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

umlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

modal logic, multiprocess networks, temporal modalities, spatial modalities

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

see reverse side

DTIC FILE COPY

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Summary

We introduce a modal logic which can be used to formally reason about synchronous fixed connection multiprocess networks such as of VLSI. Our logic has both *temporal* and *spatial* modal operators. The various temporal modal operators allow us to relate properties of the current state of a given process with properties of succeeding states of the given process. Also, the spatial modal operators allow us to relate properties of the current state of a given process with properties of the current state of neighboring processes. Many interesting properties for multiprocessor networks can be elegantly expressed in our logic. We give examples of the diverse applications of our logic to packet routing firing squad problems, and systolic algorithms.

We show that deciding validity of a formula in our logic is not decidable. However, we show that deciding validity of a proportional formula in our logic with respect to a given finite network is PSPACE-complete. We also investigate the decidability issues of different versions of this logic.

1. Introduction. One of the fundamental models of parallel computation is a collection of synchronous processors with fixed inter-connections. For example, the iterative linearly connected, mesh connected, and multidimensional arrays of [Ko69] and [Co69], the shuffle exchange networks of [St71] and ultracomputer of [Sc80], and the cube connected cycles networks of [PV79].

Parallel algorithms for such networks are difficult to formerly describe and prove correct. For example, the systolic algorithms of [KL80] are not formally proved correct in this paper; instead they present informal "picture proofs."

An informal description of a program or algorithm for a fixed connection network would likely make reference to the spatial relationships between neighboring processes and properties holding for all processes, as well as the transformations over time. Indeed, natural English allows expression of spatial modal operators such as "everywhere", "somewhere", "across such and such connection", as well as temporal modal operators such as "until", "eventually", "hereafter", and "next-time". However, natural English cannot suffice for formal semantics. This paper proposes a formal logic allowing use of these modal operators in the context of a fixed connection network. Section 2 defines our logic's syntax and semantics.

Previous program logics contained only temporal modal operations [Pn77], [MP81] or modal operations for the effect of program statements [FL79]. Temporal logic has been used to reason about parallel programs; however it is impractical to use this logic to reason about large number of processes operating synchronously and communicating through fixed connections. Our use of spatial as well as temporal modal operators is a new idea. (Note: our spatial modal operators differ in an essential way from the model operators of dynamic logic; see Section 2.3). This combination of temporal and spatial modal operators allow us to formally reason about computations on networks with complex connections.

The contribution of this paper is more than simply the definition of our network; we also describe applications and investigate its computational complexity of its decision problems.

Section 3 describes some interesting applications of our logic to routing on the shuffle exchange network, to the firing squad problem on a linear array, and to stystolic computations on arrays. We felt these examples to multiprocess networks illustrate the general applicability.

Section 4 investigates the problem of fasting validity of formulae of our logic. We show the set of valid formulas are Π_1^1-complete. However, in practice we are generally only interested in deciding validity of a proportional formula with respect to a given finite network. We show this problem is PSPACE-complete. Also, we show in the full paper that it is decidable to test validity of proportional formulae with restricted modalities (for example formulae with all temporal operators, but only the "somewhere" spatial operator, and also formulae with all spatial operators, but only the "eventually" temporal operator).

We conclude in Section 5 with a summary of our results.

2. Definition of Our Logic. We begin by describing our logic for linear time. The end of this section briefly sketches how this logic can be extended to first order formulae, and to branching time.

2.1. Networks. Let L be a countable set of symbols, which we call *links*. A *network* $G = (P,E)$ contains a countable set of *processes* P and a partial mapping $E: L \times P \to P$. For each process $p \in P$ and label $\ell \in L$, $E(\ell,p)$ is (if defined) the *process connected to* p *by link* ℓ. For example, a square grid network might have links *up*, *down*, *left*, and *right*. The links are different from atomic programs of PDL due to the restrictions given in the next page.

2.2. Syntax of the Logic. We distinguish as *temporal* modal operators the symbols *eventually, hereafter, until,* and *nexttime*. The *spatial* modal operators are *somewhere, everywhere,* and any symbol in the set of links L, which we assume contains none of the previously mentioned modal operators.

Let $\mathcal{F}_0$ be an infinite set of *atomic formulae*. Let the set of formulae $\mathcal{F}$ be the minimal set of strings containing $\mathcal{F}_0$ and such that if $f_1, f_2 \in \mathcal{F}$ then

$f_1 \wedge f_2 \in \mathcal{F}$

$\neg f_1 \in \mathcal{F}$

eventually $f_1 \in \mathcal{F}$

hereafter $f_1 \in \mathcal{F}$

f_1 *until* $f_2 \in \mathcal{F}$

nexttime $f_1 \in \mathcal{F}$

somewhere $f_1 \in \mathcal{F}$

everywhere $f_1 \in \mathcal{F}$

$\ell\ f_1 \in \mathcal{F}$ for each link $\ell \in L$

2.3. Semantics of Our Logic. Let a *model* $\mathcal{M}$ be a 5-tuple $(S, \Psi, \Delta, G, \pi)$ where:

(i) S is the set of *states*,

(ii) $\Psi: S \to 2^{\mathcal{F}_0}$,

(iii) $\Delta: (L \cup \{\textit{nexttime}\}) \times S \to S$, is a partial function

(iv) $G = (P,E)$ is a network, and

(v) $\pi: S \to P$.

Thus for each state $s \in S$, $\Psi(s)$ is the set of atomic formulas which hold at s, and $\pi(s)$ is the process associated with state s. Also, $\Delta(\textit{nexttime}, s)$ is the state occurring in the time instance just after state s, and $\Delta(\ell,s)$ is the current state of the process connected to process $\pi(s)$ by link ℓ.

We extend Δ as a partial mapping to the domain $(L \cup \{\textit{nexttime}\})^* \times S$ so that for all $s \in S$ $\Delta(\varepsilon,s) = s$, and $\Delta(\ell_1 \circ \ell_2, s)$ is defined iff $\Delta(\ell_1,s)$ and

$\Delta(\ell_2, \Delta(\ell_1,s))$ are defined and in this case $\Delta(\ell_1 \circ \ell_2,s) = \Delta(\ell_2, \Delta(\ell_1,s))$. Similarly we also extend E as a partial mapping to the domain $L^* \times P$.

A model $\mathcal{M}$ is *proper* iff

<u>R1</u>: For each link $\ell \in L$ and each state $s \in S$, $\Delta(\ell \circ \textit{nexttime},s) = \Delta(\textit{nexttime} \circ \ell, s)$ (thus *nexttime* commutes with respect to each link; this presumes the processes are synchronous).

<u>R2</u>: For each state $s \in S$, $\Delta(\textit{nexttime},s)$ is defined and $\pi(s) = \pi(\Delta(\textit{nexttime}, s))$ (thus the name of each process is invariant over time).

<u>R3</u>: For each state $s \in S$ and link $\ell \in L$, $E(\ell, \pi(s))$ is defined iff $\Delta(\ell, s)$ is defined and in this case, $E(\ell, \pi(s)) = \pi(\Delta(\ell, s))$ (thus processes associated with states are connected by the same links as in the network G)

<u>R4</u>: For any $\alpha,\alpha' \in L^*$ and states $s,s' \in S$ if $E(\alpha,\pi(s))$, $E(\alpha',\pi(s'))$ are defined and $E(\alpha,\pi(s)) = E(\alpha',\pi(s'))$ then $\Delta(\alpha,s) = \Delta(\alpha',s')$. (thus the relationship between the states of two processes is independent of the particular paths of links over which they are connected.)

<u>R5</u>: If $\pi(s_1) = \pi(s_2)$ then for some $i \geq 0$ $\Delta(\textit{nexttime}^i, s_1) = s_2$ or $\Delta(\textit{nexttime}^i, s_2) = s_1$.

Hereafter, we consider only proper models.

Let us fix the model $\mathcal{M}$. We define truth of a formulae at a given state $s \in S$ by structural induction.

For each atomic formula $F \in \mathcal{F}_0$, $s \models F$ iff $F \in \Psi(s)$. For any formulas $f_1, f_2 \in \mathcal{F}$,

$s \models f_1 \wedge f_2$ iff $s \models f_1$ and $s \models f_2$

$s \models \neg f_1$ iff $s \not\models f_1$

$s \models \textit{nexttime } f_1$ iff $\Delta(\textit{nexttime}, s) \models f_1$

$s \models \textit{eventually } f_1$ iff $\exists k \geq 0$ $\Delta(\textit{nexttime}^k, s) \models f_1$

$s \models \textit{hereafter } f_1$ iff $\forall k \geq 0$, $\Delta(\textit{nexttime}^k, s) \models f_1$

$s \models f_1$ *until* f_2 iff $\exists k \geq 0$ $\Delta(\textit{nexttime}^k, s) \models f_2$ and

$\forall i,\ 0 \leq i < k,\ \Delta(\textit{nexttime}^i, s) \models f_1$

$s \models \ell f_1$ iff $\Delta(\ell,s)$ is defined and $\Delta(\ell,s) \models f_1$

$s \models$ *somewhere* f_1 iff $\exists\alpha \in L^*$, such that $\Delta(\alpha,s)$ is defined and $\Delta(\alpha,s) \models f_1$

$s \models$ *everywhere* f_1 iff $\forall\alpha \in L^*$ $(\Delta(\alpha,s)$ is defined $\Rightarrow \Delta(\alpha,s) \models f_1)$

We let $\models_{\mathcal{M}}$ denote truth with respect to a given model $\mathcal{M}$.

2.4. Decision Problems. Formula $f \in \mathcal{F}$ is *satisfiable (valid)* if $s \models_{\mathcal{M}} f$ for some (all, respectively) model $\mathcal{M}$ and state s. Given a network G, formula $f \in \mathcal{F}$ is *G-satisfiable (G-valid)* if $s \models_{\mathcal{M}} f$ for some (all,respectively) models $\mathcal{M}$ and state s with given network G.

2.5. Extensions to a First Order Logic. The first order version of this logic consists of the additional symbols like local variables, global variables, constant symbols, function and relation sysbols, and the universal quantifier $\forall$. A term is defined as in the case of first order predicate calculus. An atomic formula is an atomic proposition or of the form $R\, t_1 t_2 \ldots t_k$ where R is k-any relation symbol (R can be equality in which case $k = 2$). The additional requirement for the set of formulae is that if f is a formula and x is a global variable so is $\forall x(f)$. A model $\mathcal{M}$ is a 5-tuple $(\Sigma, S, \Delta, G, \pi)$ where $\Sigma = (D, \alpha, \beta)$ in which D is a countable domain in which the variables take values, α interprets relation and function symbols, β is a mapping associating with each global variable and constant symbol a value from the domain; S is the set of states where each state is a mapping that associates a truth value with each atomic proposition and a value from D with each local variable; Δ, G, π are the same as in the propositional case. A proper model should satisfy the same conditions as for propositional case, modified in a natural way. We consider only proper models. Truth of an atomic formula in a state of a model is defined as in the case of first order predicate calculus; and truth of a formula in a state of a model is defined in 'uctively as in the propositional version with the following addition;

$\mathcal{M}, s \models \forall x\, f$ iff for each $c \in D$ $\mathcal{M}^c, s \models f$ where $\mathcal{M}^c$ is exactly same as $\mathcal{M}$ except that the global variable x is given the value c in $\mathcal{M}^c$. Satisfiablity and validity of formulae are defined as usual.

2.6 Extensions to a Branching Time Logic. We can easily extend our logic to a branching time logic, as in [BMP81].

3. Applications

This section gives some examples of the use of our logic to various multi-process network applications.

3.1 Routing on a Shuffle-Exchange Network.

A Shuffle-Exchange network $G = (P,E)$ where $P = \{0,1\}^n$ and

E: $\{exchange, shuffle\} \times P \to P$

is defined as follows:

$$E(exchange, (a_{n-1}, a_{n-2}, \ldots, a_0)) = (a_{n-1}, a_{n-2}, \ldots, \bar{a}_0)$$
$$E(shuffle, (a_{n-1}, a_{n-2}, \ldots, a_0)) = (a_0, a_{n-1}, \ldots, a_1)$$

for all $a_{n-1}, a_{n-2}, \ldots, a_0 \in \{0,1\}$.

Intuitively, the exchange edge connects processes p_1 and p_2 if all the bits of p_1 and p_2 are the same excepting the least significant bits which are distinct. The shuffle edge connects two processes p_1 and p_2, if p_2 is obtained by one cyclic shift of bits in p_1.

The routing problem in this network is to route a packet present at some process to a given destination traversing only along the shuffle and exchange edges.

We capture the name of a process by the atomic propositions $A_{n-1}, A_{n-2}, \ldots, A_0$. The formula f_0 asserts that the name of a process is invariant over time;

$$f_0 = \bigwedge_{0 \le i < n} (\textit{hereafter } A_i \vee \textit{hereafter } \neg A_i)$$

f_1, f_2 assert that exchange and shuffle edges are properly connected.

$$f_1 = \bigwedge_{1 \le i < n} (A_i \leftrightarrow \mathit{exchange}\ A_i) \wedge A_0 \leftrightarrow \mathit{exchange}\ \neg A_0$$

$$f_2 = \bigwedge_{0 \le i < n} (A_i \leftrightarrow \mathit{shuffle}\ A_{(i-1) \bmod n})$$

The presence of the packet at any process will be indicated by the atomic proposition X, and the destination by $D_{n-1}, D_{n-2}, \ldots, D_0$. We assume that the name of the destination travels with the message.

$$g_0 = X \wedge \bigwedge_{0 \le i < n} (A_i \supset \mathit{everywhere}\ (X \supset A_i)) \wedge (\neg A_i \supset \mathit{everywhere}\ (X \supset \neg A_i))$$

asserts that X is true at at most one place.

$$g_0 = X \wedge \bigwedge_{0 \le i < n} (D_i \supset \mathit{hereafter\ everywhere}\ (X \supset D_i)) \wedge (\neg D_i \supset \mathit{hereafter\ everywhere}\ (X \supset \neg D_i))$$

asserts that the name of the destination process travels with the packet.

$$g_2 = X \supset \mathit{nexttime}\ (X \vee (\mathit{shuffle}\ X) \vee \mathit{exchange}\ X)$$

asserts that the packet travels along shuffle or exchange edges only.

The main correctness property is g_3 which asserts that the packet reaches its destination eventually.

$$g_3 = X \wedge \bigwedge_{0 \le i \le n-1} (A_i \leftrightarrow D_i)$$

Let r be a formula which describes the actual routing algorithm. Then

$$(\mathit{hereafter\ everywhere}\ (r \wedge f_0 \wedge f_1 \wedge f_2 \wedge g_0 \wedge g_1)) \supset \mathit{eventually\ somewhere}\ g_3$$

is a valid formula iff the algorithm correctly routes packets.

Next we describe a specific routing algorithm for the shuffle exchange network and derive the corresponding formula r for its semantics. The packet will be routed in n stages, where for $i = 0, \ldots, n-1$, if at the start of the i-th stage the packet is located at a process whose lowest order address bit is not the value of D_i, then the product traverses an *exchange* link. In either case, the product next traverses a *shuffle* link and reaches the $i+1$ stage.

To define a formula r for this routing algorithm, it is useful to introduce proportional variables $S_0,\dots,S_{n-1}$ and require that only unique S_i be true at any processes, and that the S be invariant or traversing an *exchange* link but that $S_{(i+1)\bmod n}$ be true on traversing a *shuffle* link. Thus we let

$$r_0 = \bigvee_{0\leq i<n} (S_i \wedge (\bigwedge_{\substack{0\leq j<n \\ i\neq j}} \neg S_j) \wedge (\textit{nexttime exchange } S_i) \wedge (\textit{nextime shuffle } S_{(i+1)\bmod n}))$$

The formula for semantics of this routing algorithm is therefore:

$$r_1 = r_0 \wedge (X \supset \bigvee_{0\leq i<n} (S_i \wedge ((A_0 \leftrightarrow D_i) \supset \textit{nexttime exchange}(X)) \wedge ((A_0 \leftrightarrow \neg D_i) \supset \textit{nexttime shuffle}(X)))\ .$$

3.2 <u>The Firing Squad Problem for a Linear Array</u>. We briefly describe the problem and show how its correctness can be specified by our logic. A solution to the firing squad problem consists of a linear array of deterministic finite state processes as shown in figure 1. The next move of each process is a function of its present state and the states of its neighbors. All the privates are identical processes. The problem is to obtain the program for the lieutenant, the sergeant and the privates so that when even the lieutenant is in a designated initial state, then eventually all the processes simultaneously enter a special state called the firing state, and non of them enters this state before this time. The solution should work for linear arrays of *all sizes*.

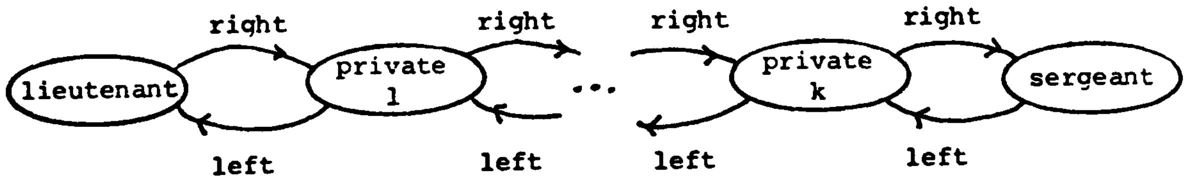


Figure 1

We assume that all processes have states sets $Q = \{0,1,2,\dots,m\}$, and the state 0 is the initial state of each process. State 1 is the specific state into which the lieutenant enters to start the operation, state m is the firing state. All the privates are identical. We use atomic propositions $P_0, P_1,\dots,P_m$ to indicate the state of an process (P_i is true at a place iff the corresponding process is in a state i at that instance). Now we assert the

the operation of the system as follows.

(i) 'I' asserts that each process is in at most one state at any instant of time.

$$I = \textit{everywhere hereafter} \left[\bigwedge_{\substack{0 \le i,j \le k \\ i \ne j}} (P_i \supset \neg P_j) \right]$$

(ii) f_0 asserts that the moves of lieutenant is according to its next move partial function $\delta_0 \colon Q^2 \to Q$.

$$f_0 = \textit{everywhere}[\neg \textit{left}(\text{true}) \supset ((P_0 \vee P_1) \wedge \textit{hereafter} \bigwedge_{i,j} ((P_i \wedge \textit{right}\ P_j) \supset \textit{nexttime}\ P_{\delta_0(i,j)}))]$$

Note that $\neg \textit{left}(\text{true})$ is true only on the lieutenant, the left most processor.

(iii) Similarly let f_1, f_2 be the formulae that define the moves of all privates and the sergeant respectively. The positions of privates is identified by the truth of the formula

$$(\textit{left}(\text{true}) \wedge \textit{right}(\text{true})).$$

Note that the position of the sergeant is identified by the formula $\neg \textit{right}(\text{True})$.

(iv) Let g_0 be the formula that asserts that if any process (other than the lieutenant) and all its neighbors are in state 0 then it remains in state 0 in the next step. It is easily seen that this can also be asserted.

Now we assert that if all the above conditions are met and at any time the lieutenant enters the state 1 then all process will eventually enter the firing state simultaneously at some future instance, and none of them will be in the firing state before that instance. This is captured by the formula g.

$$g = (I \wedge f_0 \wedge f_1 \wedge g_0) \supset \textit{hereafter} [\textit{somewhere}(\neg \textit{left}(\text{true}) \wedge P_1) \supset ((\neg \textit{somewhere}\ P_m)\ \textit{until}\ (\textit{everywhere}\ P_m))]$$

g is valid on all models with linear arrays as networks iff the given solution to the firing squad problem is correct. A similar construction can be given for the firing squad problem over any given network.

3.3 Systolic Arithmetic Computations. The systolic algorithms of [KL80] are not formally proved correct in their paper; instead they present informal "picture proofs". Our logic is thus particularly useful here when extended to first order formulae (as described in Section 2.5).

We consider an interesting example of a network for matrix-vector multiplication due to [KL80]. The matrix is an infinite band matrix of bandwidth (n+1). The network architecture is shown in figure 2.

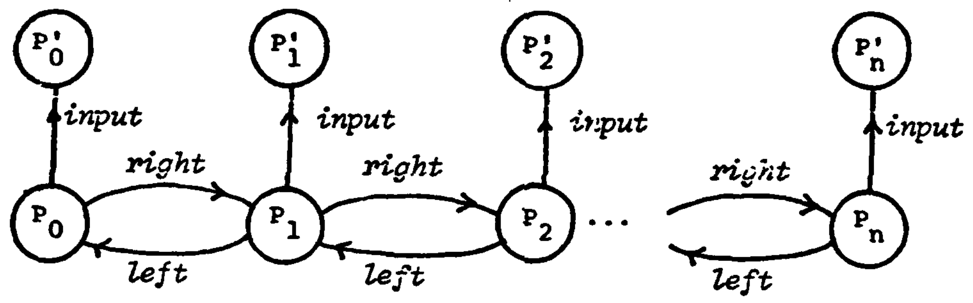


Figure 2

The main processors are $P_0, P_1, \ldots, P_n$. The processors $P'_0, P'_1, \ldots, P'_n$ are the input processors, each of them contains a variable Z. The values of Z in P'_i change with time and they represent the values of the i^{th} diagonal of the matrix. Each processor P_i has two variables X, Y. The values of the variable X in P_0 over time represent the input vector. The values of X move right with each time instance.

Thus

$$g_1 = \mathit{left}(\text{true}) \supset \forall\alpha(\mathit{left}(X=\alpha) \leftrightarrow \mathit{nexttime}(X=\alpha))$$

asserts that the value of X at the nexttime instance in a process $P_i (i > 0)$, is the present value of X in the process left to P_i.

At each step $P_i (i < n)$ computes its value of Y to be the sum of the previous value of Y in process P_{i+1}, plus the product of X in P_i times Z in P'_i. This is captured by

$$g_2 = \mathit{right}(\text{true}) \supset \forall\alpha\forall\beta(\mathit{right}(Y=\alpha) \wedge \mathit{nexttime}\ \mathit{input}(Z=\beta) \supset \mathit{nexttime}(Y = \alpha + X\cdot\beta))$$

At each step P_n computes its value of Y to be the product of the value of X in P_n and the value of Z in P'_n. This can also be easily asserted by formula

$$g_3 = \mathit{right}(\text{false}) \wedge \mathit{input}(\text{true}) \supset \forall\alpha\forall\beta(X=\alpha \wedge \mathit{input}(Z=\beta)) \supset \mathit{nexttime}(Y=\alpha\cdot\beta)).$$

(note that $\mathit{right}(\text{false}) \wedge \mathit{input}(\text{true})$ holds only for process P_n)

The correctness property at P_n can thus be expressed in our logic as

$$\mathit{hereafter\ everywhere}\ (g_1 \wedge g_2 \wedge g_3) \supset \mathit{hereafter}\ h$$

where

$$h = \mathit{left}(\mathit{false}) \wedge \mathit{input}(\text{true}) \supset$$

$$\forall\alpha_0\ldots\alpha_n\forall\beta_0\ldots\beta_n\ (\bigwedge_{i=0}^{n} \mathit{nexttime}^i(X=\alpha_i) \wedge \mathit{nexttime}^{n+i}(Z=\beta_i)) \supset \mathit{nexttime}^{2n}(Y = \sum_{i=0}^{n} \alpha_i\cdot\beta_i) .$$

4. <u>Decidability and Complexity Issues</u>. In this section we consider issues of decidability and complexity of different versions of our logic. Recall that a formula is said to be satisfiable iff there exists a model and a state at which the formula is true. A formula is said to be valid if it is true in all states of all models. We say that a formula is satisfiable (valid) on finite networks if the formula is true in a (all) model with finite networks.

THEOREM 1. *The set of satisfiable formulae of multiprocessor network logic is Σ^1_1-complete and the set of valid formulae is Π^1_1-complete.*

Proof sketch: First we show that the set of satisfiable formulae is a Σ^1_1-complete set. From this result it can easily be shown that the set of valid formulae is Π^1_1-complete.

We consider a deterministic Turing machine M on infinite strings. M has one read only infinite input tape, and an infinite work tape. An infinite string is input to M on its input tape. M never halts. M is said to accept an input if during its computation it goes into any of a set of final states infinitely often. The set of encodings of all Turing machines that accept at least one input, is shown to be $\frac{1}{1}$-complete in [SCFG82]. We reduce this set to the set of satisfiable formulae. An ID of M is the part of input is seen thus far, the contents of the work tape, the position of the head on the work tape. We define a sequence of IDs of M during its computation on an input and express this sequence using a formula in the logic. We also assert that in this sequence final IDs (IDs having a final state) appear infinitely often. Thus given an encoding of a Turing machine we obtain a formula that is satisfiable iff the Turing machine accepts at least one input. The details will be given in the full paper. □

Let $\mathcal{M} = (S, \Psi, \Delta, G, \pi)$ be a model where $G = (P,E)$ is a finite network. Let $\phi: P \to S$. ϕ is said to be consistent with $\mathcal{M}$, if $\pi(\phi(p)) = p$ for all $p \in P$, and for all p_i, p_j if $p_j = E(\ell, p_i)$ for some $\ell \in L$, then $\phi(p_j) = \Delta(\ell, \phi(p_i))$. Let $\Phi = \{\phi \mid \phi$ is consistent with $\mathcal{M}\}$, and let $next: \Phi \to \Phi$ be such that for all $\phi \in \Phi$ and for all p $next(\phi)(p) = \Delta(nexttime, \phi(p))$. $\mathcal{M}$ is said to be *ultimately periodic* with starting index ℓ and period m, if for all $\phi \in \Phi$ $next^i(\phi) = next^{i+m}(\phi)$ for all $i \geq \ell$. For any formula f, let $SF(f)$ be the set of subformulae of f, and for any $\phi \in \Phi$, let $[\phi]: P \to 2^{SF(f)}$ such that $[\phi](p) = \{g \mid g \in SF(f)$ and $\phi(p) \models g\}$. We require a technical lemma characterizing satisfiability.

LEMMA 1. f *is satisfaible in a model over a finite network iff* f *is satisfiable over an ultimately periodic model over a finite network.* □

THEOREM 2. *The set of formulae that are satisfiable in a model over a finite network is* Σ_1^0*-complete, and the set of valid formulae in models over finite networks is* Π_1^0*-complete.*

Proof: As in the previous theorem, we can reduce the halting problem of Turing machines over finite strings to the set of satisfiable formulae in a model over a finite network. We give a Turing machine M which accepts the above set. M guesses a finite network and an ultimately periodic model over this network. It next verifies that f is satisfiable in this model. M halts only on the input formulae that are satisfiable in a model over a finite network. □

THEOREM 3. *The following problem is* PSPACE*-complete. Given a finite network* G, *and a formula* f, *is* f *satisfiable in a model over the network* G?

Proof: The PSPACE-hardness of the problem follows from the PSPACE-hardness of satisfiablility for linear time temporal logic [SC82]. We give a polynomial space bounded Turing machine M that checks if f is satisfiable in a model over the network G. M guesses $[\phi]$, and verifies for consistency and that $f \in [\phi](p)$ for some $p \in P$. At each subsequent instance M guesses $[\mathit{next}(\phi)]$ and checks that it is consistent with $[\phi]$. It continues this each time keeping $[\phi]$ and $[\mathit{next}](\phi)$. At a certain instance it guesses the beginning of the period and saves the corresponding $[\phi]$. It continues the previous process, each time guessing either $[\mathit{next}(\phi)]$ or guessing that it is the end of the periodic part. In the latter case it takes $[\mathit{next}(\phi)]$ to be the saved value at the beginning of the period. Each time M guesses $[\mathit{next}(\phi)]$ it verifies that $[\phi]$ is consistent with [*next*(ϕ)]. M also verifies that certain formulae are fulfilled in the periodic part. M clearly uses space polynomial in the size of G and the size of f. Further, we can show: $\square$

THEOREM 4. *The set of valid formulae of first order multiprocessor network logic over models on finite networks is* Π_1^1*-complete.* $\square$

5. Conclusions. We have proposed a logic to reason about computations of multiprocessor networks. We feel that our logic will be useful to specify the semantics and prove correctness of multiprocess networks. No such formal system for multiprocessor networks had been proposed previously. We have examined the application of our logic to some diverse multiprocess network problems, and presented some results in decidability and complexity of our logic.

Bibliography

[BMP81] M. Ben-ari, Z. Manna, A. Pnueli, "The temporal logic of branching time", 8th ACM Symposium on Principles of Programming Languages, Williamsburg, VA, 1981.

[CE81] E. M. Clarke, A. Emerson, "Design and synthesis of programming skeletons using branching time temporal logic", IBM Conference of Logics of Programs, May 1981.

[Co69] S. N. Cole, "Real time computations by n-dimensional iterative arrays of finite state machines, IEEE Trans. on Computers, 18 (1969) pp. 349-365.

[FL79] M. Fischer, R. Ladner, "Propositional dynamic logic of regular programs", JCSS, 18(2), 1979.

[GPSS] D. Gabbay, A. Pnueli, S. Shealah, J. Stavi, "Temporal analysis of fairness", 7th ACM Symposium on Principles of Programming Languages, Las Vegas, NV.

[HR81] J. Y. Halpern, J. H. Reif, "The propositional dynamic logic of deterministic, well-structured programs", 22nd Symposium on Foundations of Computer Science, Nashville, TN, 1981.

[MP81] Z. Manna, A. Pnueli, "Verification of concurrent programs", The Correctness Problem in Computer Science, Academic Press, London, 1981.

[Ko69] S. R. Kosaraju, "Computations on iterative automata", Ph.D. Thesis, University of Pennsylvania, PA, 1969.

[Pn77] A. Pnueli, "The temporal logic of programs", Proceedings of 18th Symposium on Foundations of Computer Science, Providence, RI, Nov. 1977.

[PV79] F. P. Preparata, J. Vuillemin, "The cube connected cycles: a versatile network for parallel computation", FOCS 1979, pp. 140-147.

[Sc80] J. T. Schwartz, "Ultracomputers", ACM Trans. on Programming Languages and Systems, Vol. 12, No. 4, Oct. 1980, pp. 484-521.

[SCFG82] A. P. Sistla, E. M. Clarke, N. Francez, Y. Gurevich, "Are message buffers characterizable in linear temporal logic?", Proceedings of the Symposium on Principles of Distributed Computing, Ottawa, Canada, August 1982.

[SC82] A. P. Sistla, E. M. Clarke, "The complexity of propositional linear temporal logics", ACM Symposium on Theory of Computing 1982, pp. 159-167.

[St71] H. S. Stone, "Parallel processing with the perfect shuffle", IEEE Trans. on Computers, Vol. C-20, No. 2, Feb. 1971, pp. 153-161.